



Università degli studi di Parma

Dipartimento di Ingegneria e Architettura

Sistemi operativi e in tempo reale - a.a. 2022/23

Dai Processi alla Programmazione Concorrente



- ❑ La natura sequenziale o non sequenziale dei problemi e delle nostre soluzioni
- ❑ La componente dinamica del processo: in questa lezione la chiameremo talvolta *processo* (denominazione tradizionale), anziché *thread* (terminologia attuale), in omaggio all'epoca in cui queste idee sono state sviluppate
- ❑ Rappresentazione della esecuzione di un processo: eventi, traccia
- ❑ Classificazione delle situazioni e dei problemi



- ❑ In un sistema multiprogrammato la CPU esegue alternativamente, in un intervallo di tempo, *sequenze di operazioni appartenenti a processi diversi*; in un dato intervallo l'esecuzione di un processo può essere *sospesa e ripresa più volte*
- ❑ A differenza dei sistemi monoprogrammati, nei sistemi multiprogrammati occorre distinguere tra l'attività della CPU e l'esecuzione di un particolare processo



- ❑ In un sistema multiprogrammato *sono presenti contemporaneamente più processi/thread*
- ❑ Se il sistema hw dispone di una sola CPU, *un solo* processo in ogni istante può essere in *esecuzione*. Gli altri processi sono sospesi in attesa della CPU o del verificarsi di particolari condizioni che rendano possibile il proseguimento della loro esecuzione (es. completamento I/O)
- ❑ Stato di un processo/thread:
 - in esecuzione (running)
 - bloccato (idle, waiting)
 - pronto (ready)



- ❑ Il concetto di processo rappresenta una *astrazione* della sequenza di operazioni svolte per eseguire un determinato programma, indipendentemente da:
 - processore fisico utilizzato
 - tempo di esecuzione
- ❑ E' uno strumento utile nello studio di sistemi in cui più attività evolvono *contemporaneamente*, in maniera reale o simulata (sistemi multiprogrammati, controllo dei processi industriali)
- ❑ E' un'astrazione utile nella scomposizione di sistemi complessi

Livello di rappresentazione dei processi



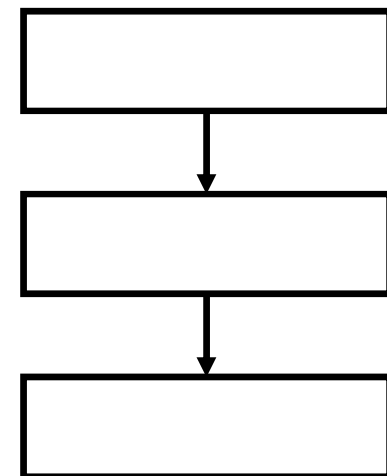
Le operazioni di un processo possono essere descritte a diversi livelli di dettaglio:

- $x := x + 1;$

- | | |
|-------|-----|
| LOAD | A,x |
| INCR | A |
| STORE | x,A |

- `ComputeNextItem(x);`

- `Im=GetImage();`
`P_Im=ProcessImage(Im);`
`Out=RenderImage(P_Im);`



Gestione dei processi



- ❑ La CPU può essere *commutata in un qualsiasi istante* da un processo ad un altro
- ❑ \Rightarrow è indispensabile *ad ogni commutazione salvare tutte le informazioni contenute nei registri della CPU* e relative al processo/thread che è stato sospeso (PC, accumulatori, registri indice, etc.)
- ❑ *Descrittore di processo o thread, ovvero process control block (PCB) o thread control block (TCB):* area di memoria, mantenuta entro l'area protetta del SO, associata al processo/thread e contenente tutte le informazioni proprie del processo/thread

Algoritmo, Programma, Processo



- ❑ *Algoritmo*: Procedimento logico che deve essere seguito per risolvere il problema in esame
- ❑ *Programma*: Descrizione dell'algoritmo tramite un opportuno formalismo (*linguaggio di programmazione*) che rende possibile l'esecuzione dell'algoritmo da parte di un particolare elaboratore
- ❑ *Processo (sequenziale)*: La sequenza di eventi cui dà luogo un elaboratore quando opera sotto il controllo di un particolare programma (evento = esecuzione di una operazione)



Rappresentazione di un processo

- Sequenza di *stati* attraverso i quali "passa" l'elaboratore durante l'esecuzione di un programma (storia del processo, traccia della esecuzione del programma)
- Esempio: Algoritmo di Euclide per il calcolo del M.C.D. di x e y (numeri naturali)

begin

$a := x; b := y;$

while ($a \neq b$) *do*

if $a > b$ *then* $a := a - b$

else $b := b - a$

end



Rappresentazione di un processo

x	18	18	18	18	18	18
y	24	24	24	24	24	24
a	-	18	18	18	12	6
b	-	-	24	6	6	6

- ❑ Lo stato è espresso dai valori delle variabili (oltre che dallo stato dei registri)
- ❑ Il processo può essere sospeso in qualsiasi stato



Un programma -> più processi

- ❑ Un programma descrive non un processo ma *un insieme* (potenzialmente infinito) *di processi*, ognuno dei quali è relativo all'esecuzione del programma da parte della CPU *per un determinato insieme di dati di ingresso*
- ❑ Processi che derivano dal medesimo programma possono essere attivi contemporaneamente

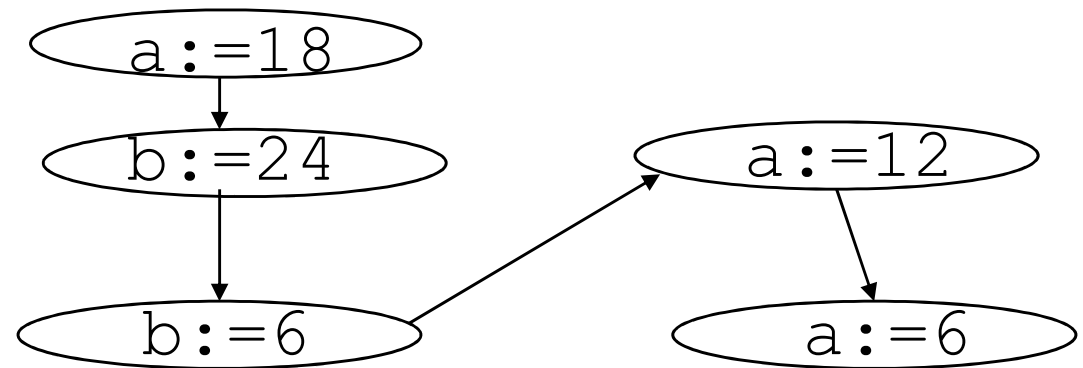


Grafo di precedenza

- ❑ Un processo può essere rappresentato tramite un *grafo orientato*, detto *grafo di precedenza* del processo
- ❑ I *nodi* del grafo rappresentano i *singoli eventi* del processo, mentre gli *archi* identificano le *precedenze temporali* tra tali eventi
- ❑ Il processo è *strettamente sequenziale* → il grafo di precedenza è ad ordinamento totale (ogni nodo ha esattamente un predecessore ed un successore):

Grafo di precedenza

- Un processo può essere rappresentato tramite un *grafo orientato*, detto *grafo di precedenza* del processo
- I *nodi* del grafo rappresentano i *singoli eventi* del processo, mentre gli *archi* identificano le *precedenze temporali* tra tali eventi
- Il processo è *strettamente sequenziale* → il grafo di precedenza è ad ordinamento totale (ogni nodo ha esattamente un predecessore ed un successore):





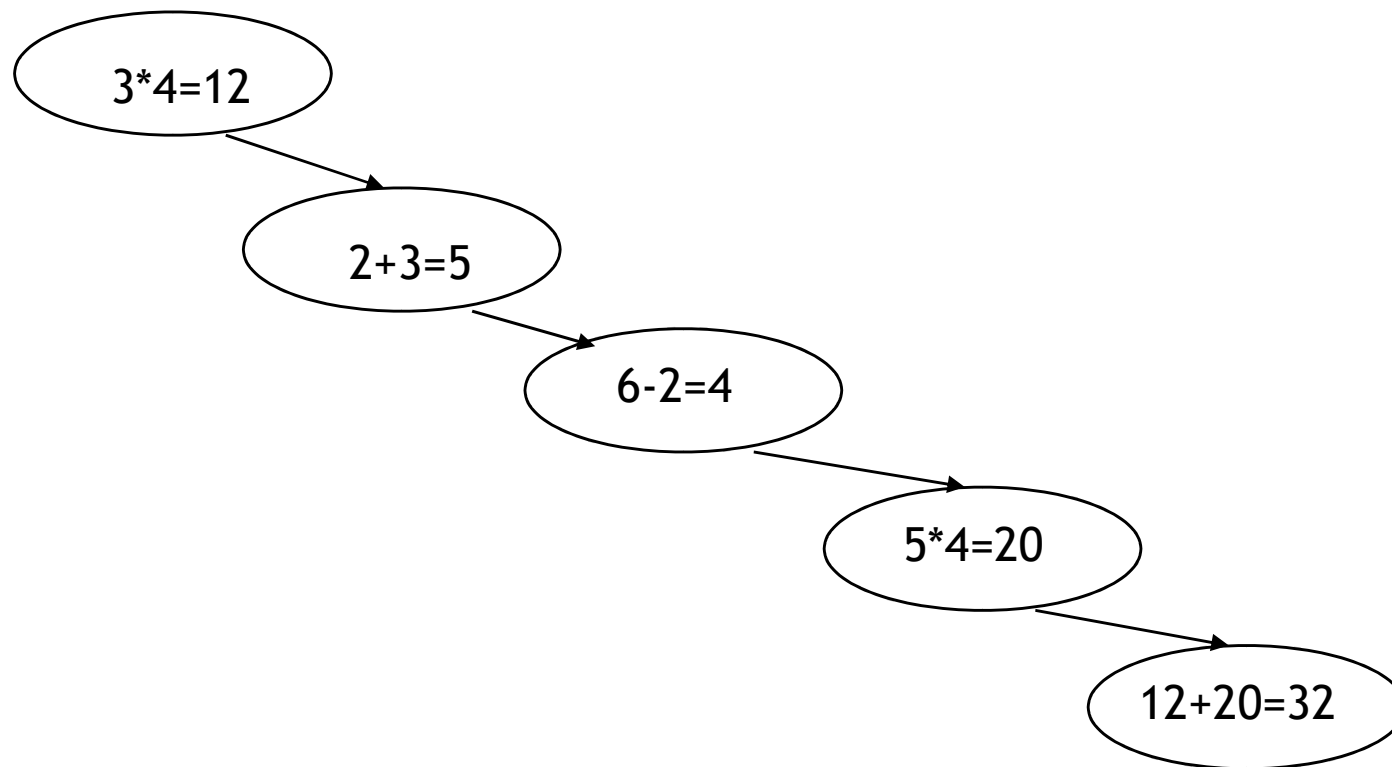
Processi non sequenziali

- ❑ L'ordinamento totale di un grafo di precedenza deriva in genere dalla *natura sequenziale* del processo (a sua volta imposta dalla natura sequenziale dell'elaboratore)
- ❑ In taluni casi l'ordinamento totale è *implicito* nel problema da risolvere. Spesso è invece una *imposizione* che deriva dalla natura sequenziale dell'elaboratore
- ❑ Molte applicazioni potrebbero più naturalmente essere rappresentate da processi (*non sequenziali*) tra i cui eventi sussiste un ordinamento non totale ma *solo parziale*



Processi non sequenziali

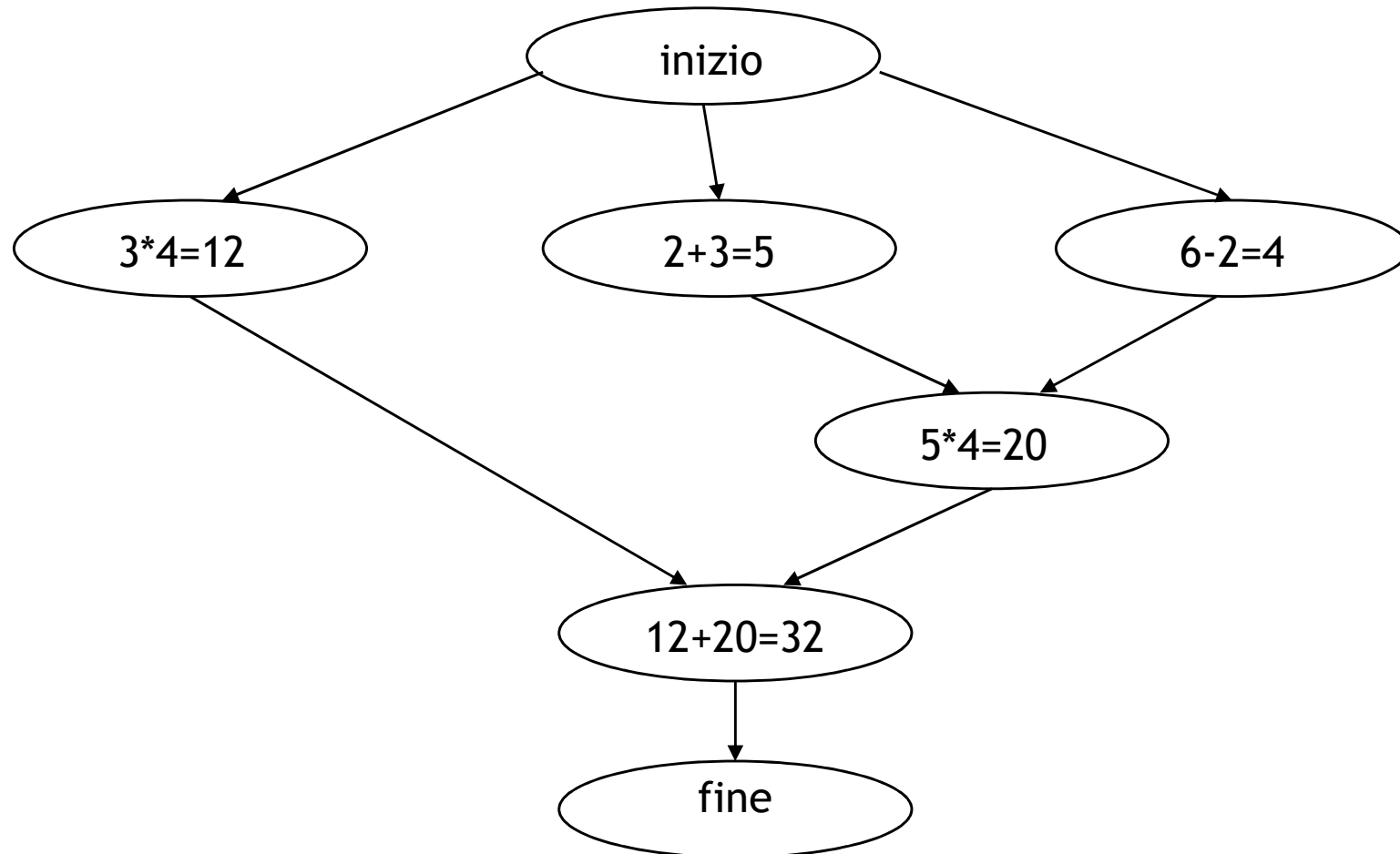
- ❑ Esempio - Valutazione della espressione:
 $(3 * 4) + (2 + 3) * (6 - 2)$
- ❑ Grafo di precedenza ad ordinamento totale:



Processi non sequenziali



- Grafo di precedenza ad ordinamento parziale:





Processi non sequenziali

- ❑ Nell'esempio precedente la logica del problema *non impone* un ordinamento totale tra le operazioni da eseguire - ad esempio: tra $(2+3)$ o $(6-2)$
- ❑ Le due operazioni tuttavia devono essere eseguite prima del prodotto dei loro risultati
- ❑ La presenza di eventi del processo tra cui non sussistono vincoli di precedenza ne denota la natura *non sequenziale*: il risultato dell'elaborazione è indipendente dall'ordine con cui si verificano tali eventi
- ❑ Non solo, i due eventi potrebbero essere eseguiti anche in parallelo
- ❑ In serie, in quale ordine, in parallelo? E' irrilevante!
 \Rightarrow *in modo concorrente*

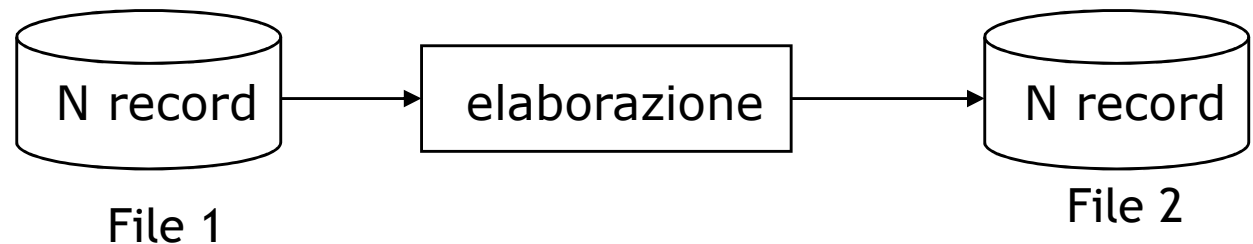


Processi non sequenziali

- Esempio: Elaborazione dati su un file sequenziale

```
var    buffer: T;  
      i: 1.. N;
```

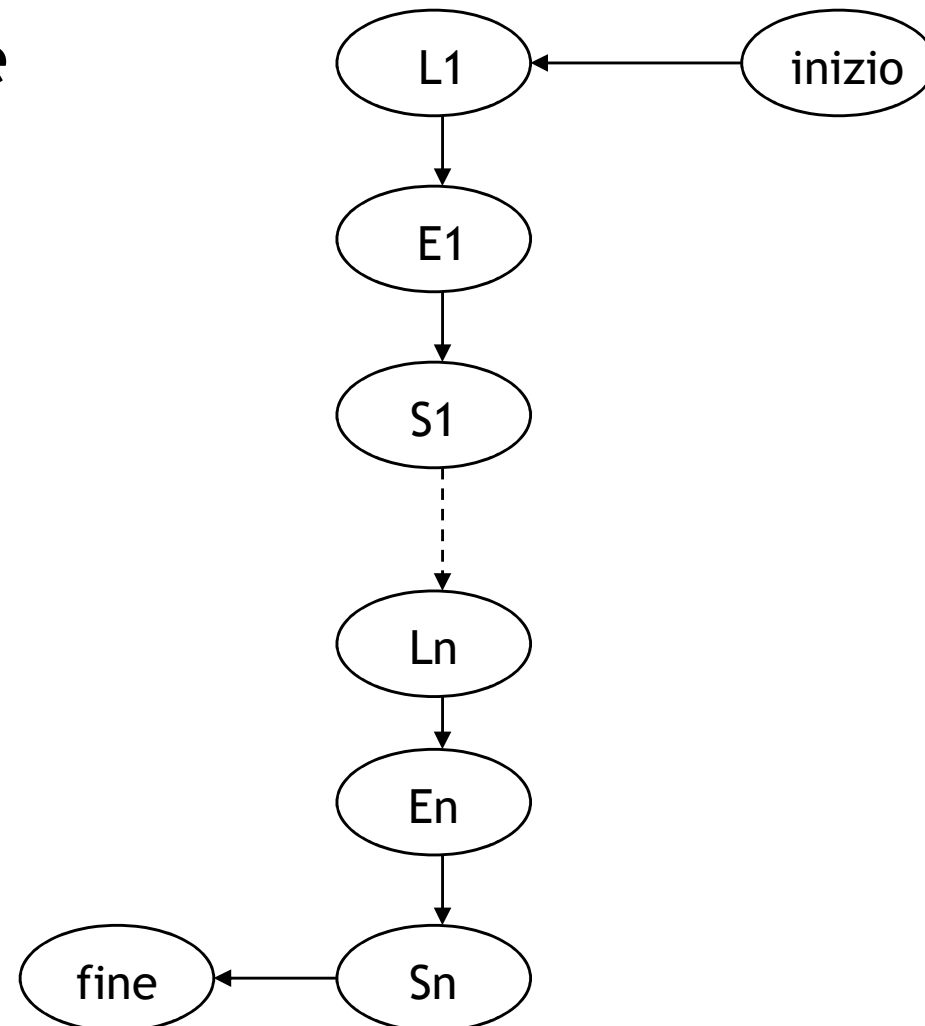
```
begin  
  for i := 1 to N do  
    begin  
      lettura (buffer);  
      elaborazione (buffer);  
      scrittura (buffer);  
    end  
  end  
end
```



Processi non sequenziali



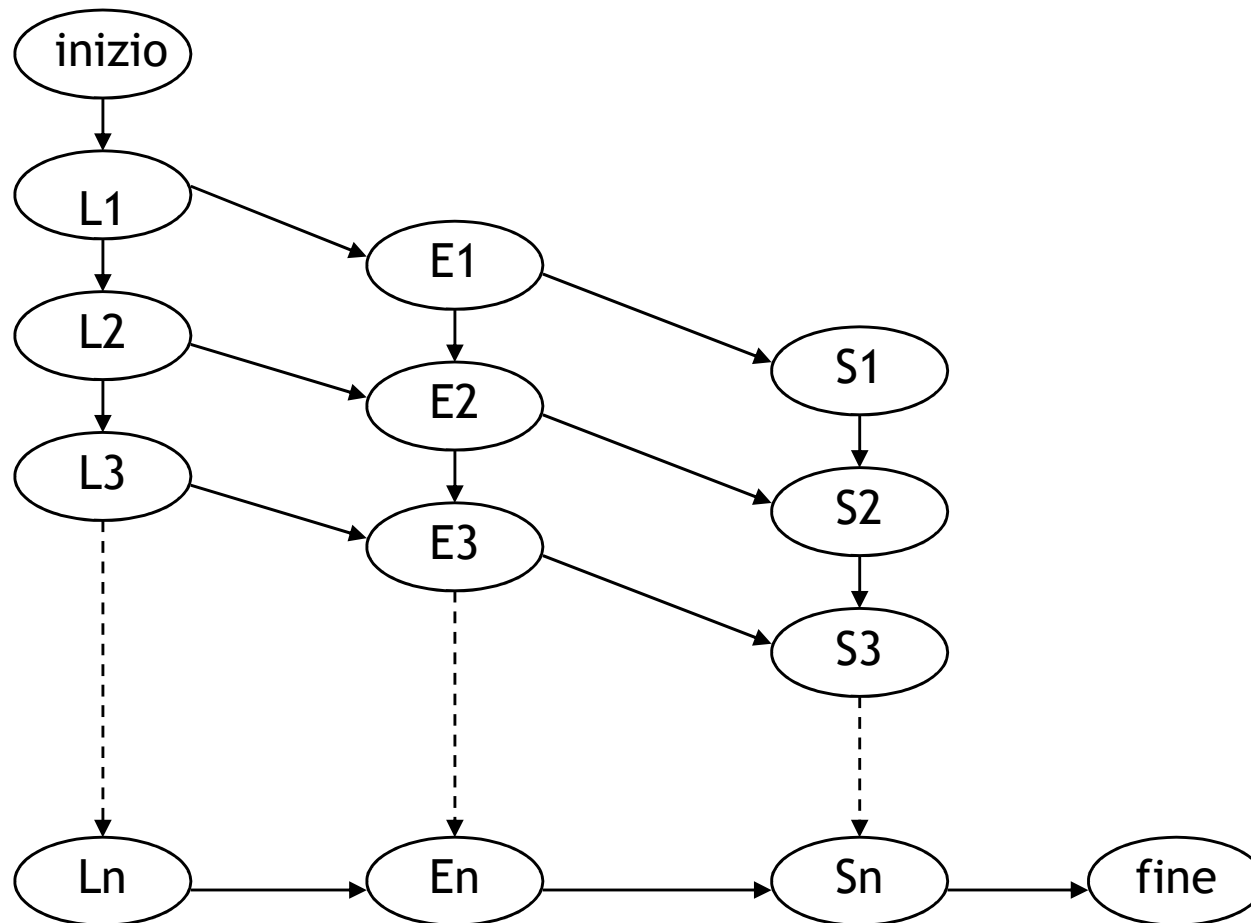
- Grafo di precedenza ad ordinamento totale



Processi non sequenziali



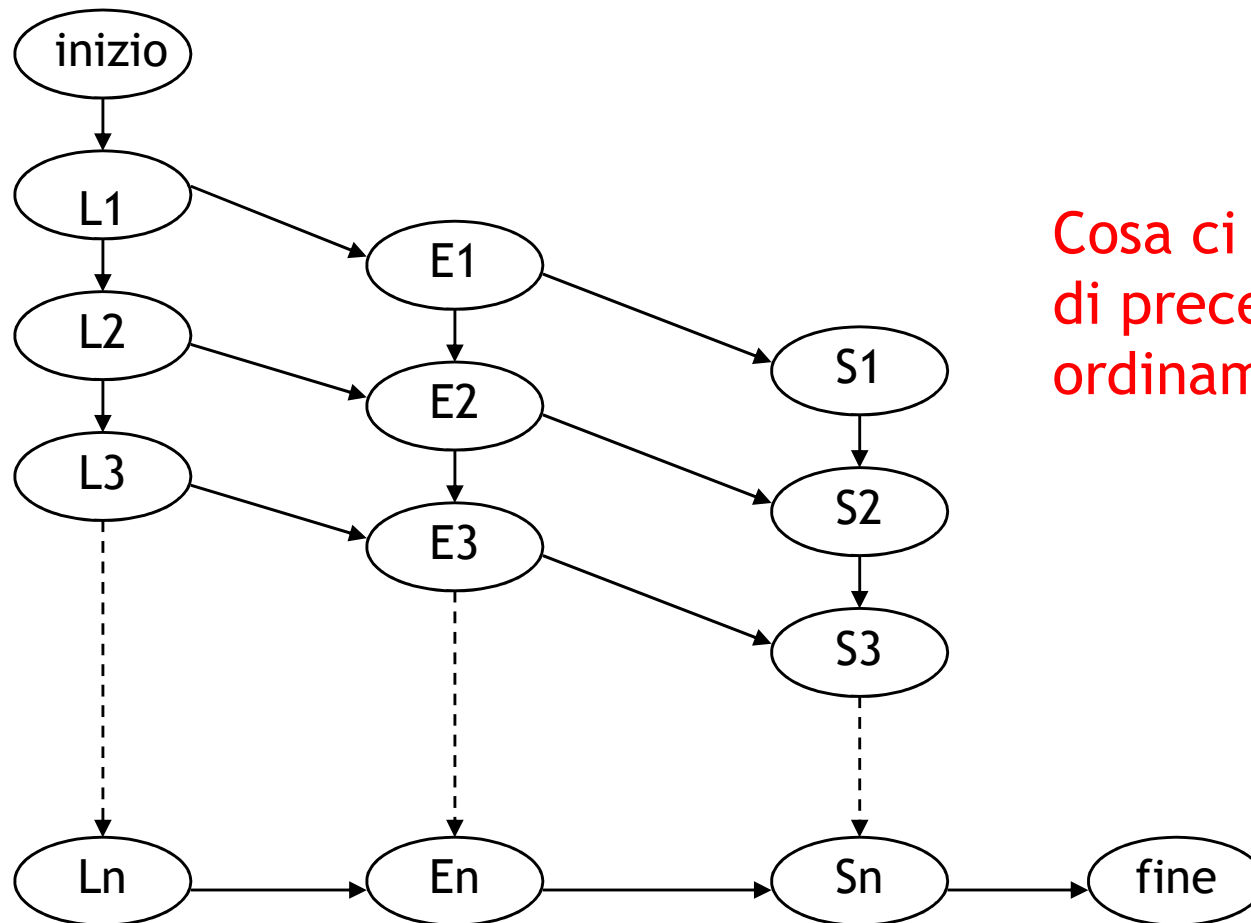
- Grafo di precedenza ad ordinamento parziale:



Processi non sequenziali



- Grafo di precedenza ad ordinamento parziale:



Cosa ci dice un grafo di precedenza ad ordinamento parziale?

Processi non sequenziali

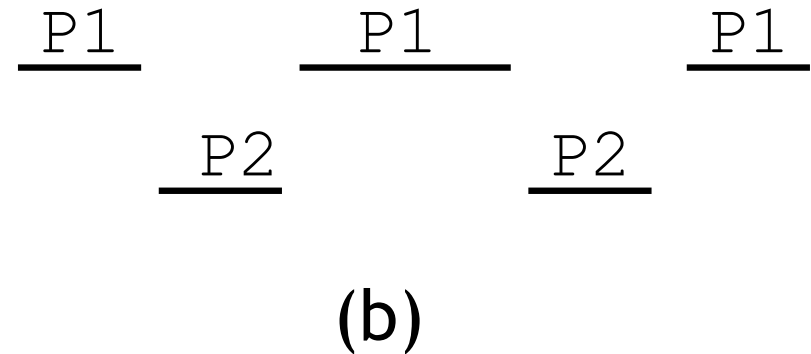
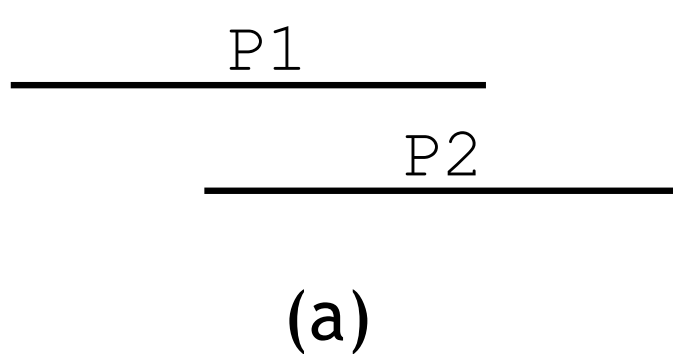


- ❑ Un grafo di precedenza ad ordinamento parziale identifica un processo non sequenziale
- ❑ La rappresentazione dell'esecuzione di un programma tramite processo non sequenziale:
 - esprime il massimo grado di *parallelismo* interno del problema (efficienza)
 - costituisce la formulazione *più naturale* del problema
- ❑ L'esecuzione di un processo non sequenziale richiede:
 - 1) elaboratore non sequenziale
 - 2) linguaggio di programmazione non sequenziale

Processi non sequenziali



- ❑ Elaboratore non sequenziale (in grado di eseguire più operazioni contemporaneamente):
 - architetture parallele e multicore (a)
 - sistemi multielaboratori (a)
 - sistemi monoelaboratore (b)





Linguaggio non sequenziale

- ❑ E' difficile per il programmatore riuscire a dominare la complessità di un algoritmo non sequenziale
- ❑ I linguaggi di programmazione non sequenziale consentono di descrivere una elaborazione non sequenziale come *composizione di processi sequenziali* eseguiti "contemporaneamente", ma analizzati e programmati *separatamente*
 - *processi (o thread) concorrenti*
- ❑ → Semplificazione, modularizzazione dello sviluppo e della comprensione del programma complessivo
- ❑ L'attività svolta dall'elaboratore dedicato di *ciascun processo sequenziale* può essere rappresentata da un grafo di precedenza ad ordinamento totale

Programmazione non sequenziale:

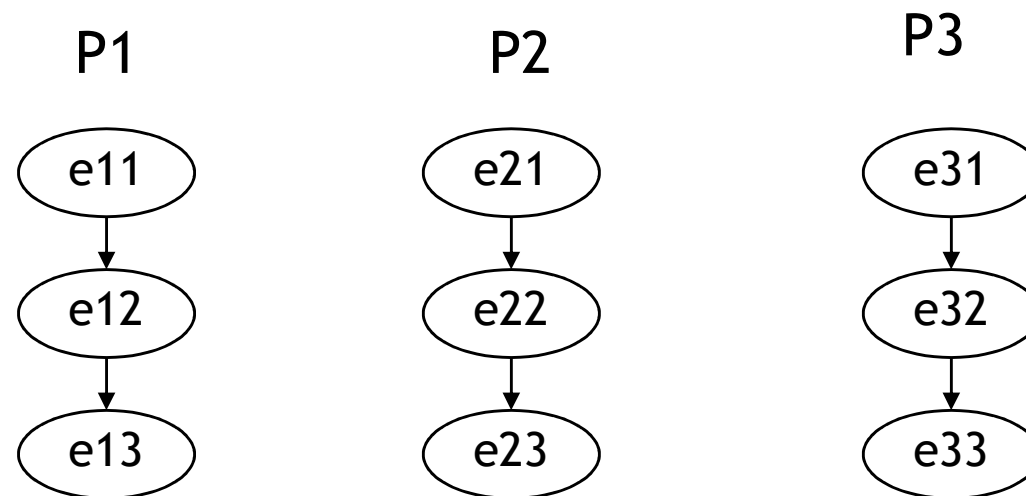
Classificazione



- ❑ Scomposizione della elaborazione di un algoritmo parallelo in *processi sequenziali concorrenti*
- ❑ Classificazione:
 - *processi indipendenti*: il grafo di precedenza è costituito da un insieme di sottografi ad ordinamento totale tra loro non connessi. Ogni sottografo identifica un processo
 - *processi interagenti*: il grafo di precedenza è un grafo connesso ad ordinamento parziale. Si individua sul grafo un insieme ($P_1 \dots P_n$) di *sequenze di nodi*, su ognuna delle quali cioè vale un ordinamento totale

Processi indipendenti

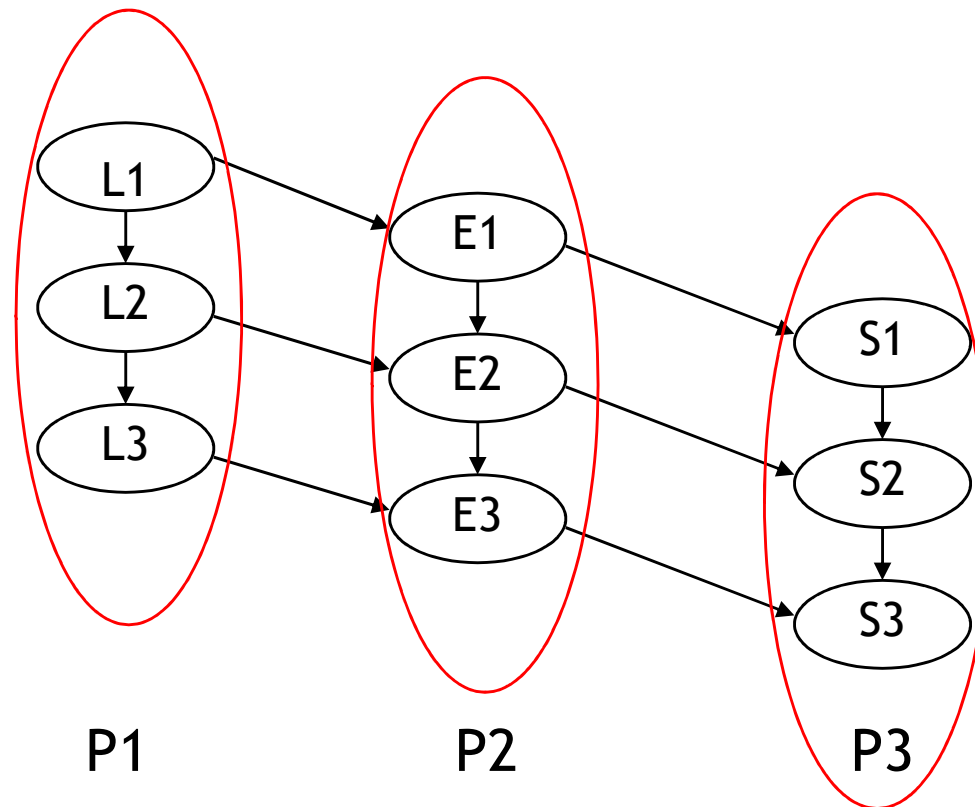
- Processi *asincroni*: la velocità relativa dei diversi elaboratori può essere totalmente diversa
- L'evoluzione di un processo *non influenza* quella dell'altro



Processi interagenti



- Esempio dell'elaborazione su nastro: processo di lettura (P1), processo di elaborazione (P2), processo di scrittura (P3)



Processi interagenti



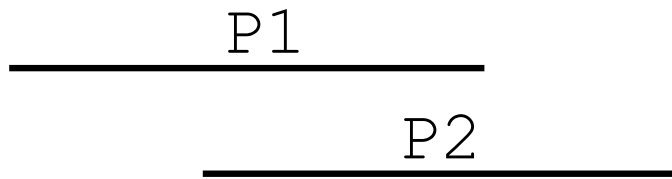
- Un *arco* che collega nodi di processi diversi rappresenta un *vincolo di precedenza* tra i corrispondenti eventi
- Affinché i processi interagenti esprimano correttamente la elaborazione non sequenziale è necessario imporre vincoli di precedenza tra le operazioni dei processi (*vincoli di sincronizzazione*); corrispondono agli archi omessi del grafo parzialmente ordinato di partenza
- Vincolo di sincronizzazione: *ordinamento di eventi*
- I processi possono essere eseguiti indipendentemente ma debbono rispettare tali vincoli
- La scomposizione in generale *non è unica*: conviene individuare un insieme di processi che dia luogo a poche interazioni



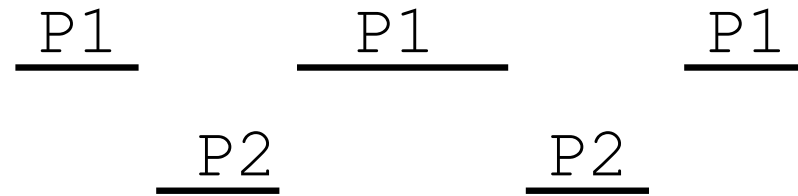
Processi concorrenti

- Due processi o thread si dicono concorrenti se la loro esecuzione *si sovrappone* nel tempo

overlapping



interleaving



- Più in generale: due processi o thread sono concorrenti se *la prima operazione di un processo / thread inizia prima dell'ultima operazione dell'altro*



Processi concorrenti

- ❑ Motivi per una esecuzione in parallelo:
 - *condivisione* di risorse fisiche
 - *condivisione* di risorse logiche
 - incremento della *velocità* di esecuzione
 - *modularità*

- ❑ La velocità di esecuzione migliora grazie alla suddivisione del task in sottotask, ciascuno dei quali eseguito in parallelo con altri:
 - presuppone la presenza di elementi multipli di elaborazione centrali e/o dedicati
 - multicore



□ Processi indipendenti:

un processo è indipendente se *non può* influenzare o essere influenzato da altri processi

□ Caratteristiche:

- il suo stato *non è condiviso* da altri processi
- la sua esecuzione è *deterministica*: il risultato della esecuzione dipende solo dallo stato di ingresso
- la sua esecuzione è *riproducibile*: il risultato dell'esecuzione è sempre lo stesso per un medesimo stato di ingresso
- la sua esecuzione può essere *bloccata e fatta ripartire* senza provocare danni

Relazioni tra processi



□ Processi interagenti:

un processo è interagente (ad es. cooperante) se può influenzare o essere influenzato da altri processi

□ Caratteristiche:

- il suo stato è *condiviso* da altri processi
- la sua esecuzione è *non deterministica*: il risultato della sua esecuzione dipende dalla sequenza di esecuzione relativa dei processi e non è predicibile
- la sua esecuzione è *non riproducibile*: il risultato non è sempre lo stesso per il medesimo stato di ingresso



Interazione tra processi

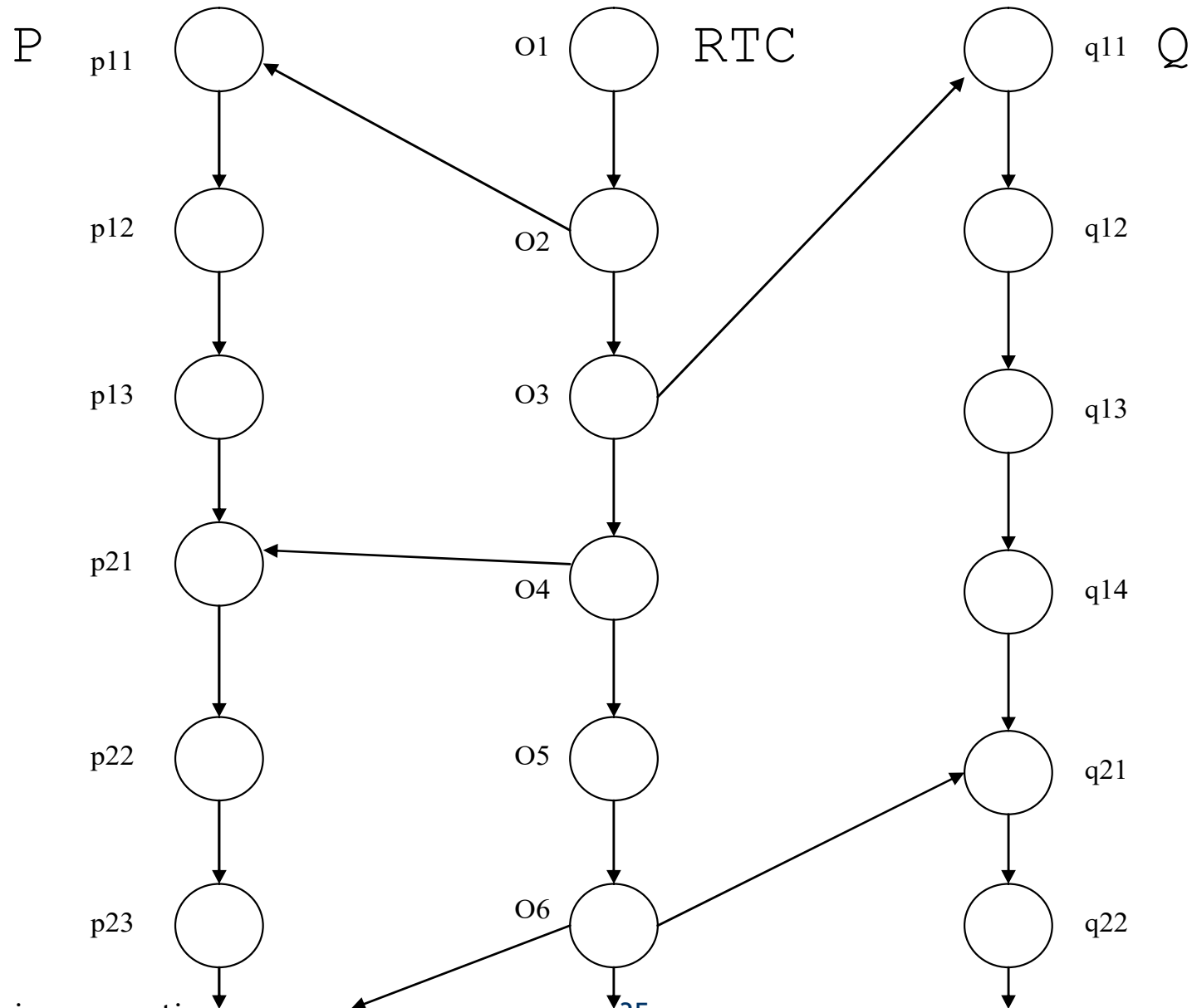
- ❑ Cooperazione:
- ❑ Comprende tutte le interazioni *prevedibili e desiderate*, insite cioè *nella logica dei programmi* (archi nel grafo di precedenza ad ordinamento parziale)
- ❑ Prevede *scambio di informazioni*:
 - segnale temporale (senza trasferimento di dati)
 - messaggi (dati)
- ❑ In entrambi i casi esiste un *vincolo di precedenza* (*sincronizzazione*) tra gli eventi di processi diversi; nel secondo caso è presente anche una *comunicazione* di dati tra i processi



Processi cooperanti

- ❑ Esempio: processo orologio (Real-Time Clock)
- ❑ *Cooperazione* con scambio di segnali temporali
 - Problematica tipica dei sistemi real-time per il controllo di processo

Esempio di cooperazione: Processo orologio (real-time clock)





Interazione tra processi

- ❑ Competizione:
- ❑ La "macchina concorrente" su cui i processi /thread sono eseguiti mette a disposizione un *numero limitato di risorse*
- ❑ → Competizione per l'uso di risorse comuni che *non possono essere usate contemporaneamente*
- ❑ Interazione *prevedibile e non desiderata ma necessaria*

Processi in competizione

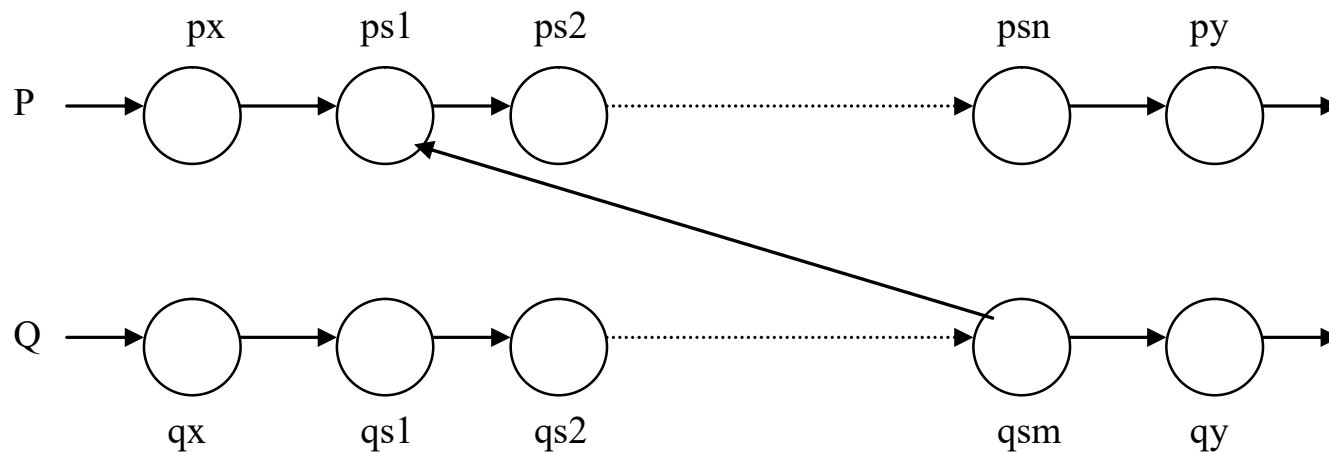
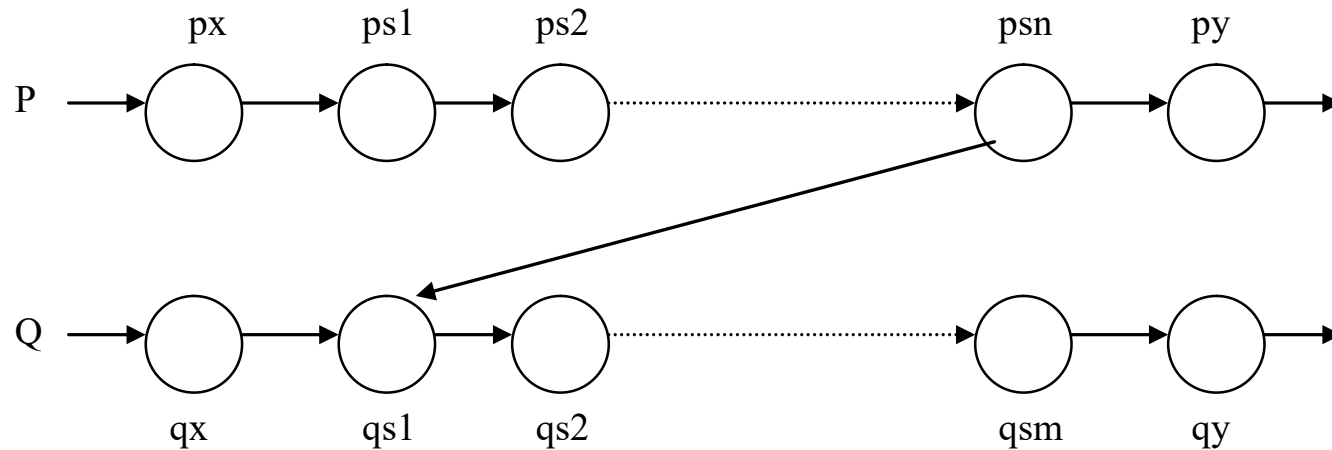


- Esempio: accesso ad un dispositivo di output condiviso
 - Es. Telescrivente / Teletype / TTY
 - (archeologia informatica!)



- *Competizione* per accedere al dispositivo in maniera mutuamente esclusiva
- Due processi P e Q includono *sequenze di eventi* ($ps_1, ps_2, \dots ps_n$) e ($qs_1, qs_2, \dots qs_m$) che non possono essere eseguite in maniera concorrente tra loro, ma non è necessario un particolare ordinamento tra le sequenze

Esempio di competizione: Accesso a dispositivo di output condiviso



Interazione tra processi



- ❑ Cooperazione => Sincronizzazione diretta o esplicita
- ❑ Competizione => Sincronizzazione indiretta o implicita

- ❑ Esistono altre forme di interazione?

Interazione tra processi



- ❑ Interferenza
- ❑ Provocata da *errori di programmazione*:
 - inserimento nel programma di interazioni tra processi non richieste dalla natura del problema
 - erronea soluzione a problemi di interazione (cooperazione e competizione) necessari per il corretto funzionamento del programma
- ❑ Interazione *non prevista e non desiderata*
- ❑ Dipende dalla *velocità relativa* tra i processi: "gli effetti possono o meno manifestarsi nel corso dell'esecuzione del programma a seconda delle diverse condizioni di velocità di esecuzione dei processi" (*errori dipendenti dal tempo*)

Il quarto caso



- ❑ Interazione non prevista ma desiderata

- ❑ Programmazione evolutiva?
- ❑ Caos primordiale?
- ❑ Biglietto della lotteria?



- ❑ La legge (empirica) di Murphy suggerisce che interazioni di questo tipo siano poco plausibili: interazioni impreviste tra thread/processi si traducono quindi in *interferenze*!



Processi interferenti

- ❑ Problema fondamentale della programmazione concorrente: *eliminazione delle interferenze*
- ❑ L'eliminazione delle interferenze del primo tipo è semplificata se la macchina concorrente fornisce *meccanismi di protezione degli accessi*
 - *spazio di indirizzamento, concetto di processo*
- ❑ Le interferenze del secondo tipo sono *previste* ma programmate in modo errato → la protezione degli accessi non risolve!
- ❑ Servono tecniche di *programmazione concorrente strutturata*
 - *Programmazione multithread*



Pattern di programmazione concorrente

- ❑ Il SO, tramite scheduler e dispatcher, può scegliere e mettere in esecuzione i thread / processi in un ordine arbitrario...
- ❑ I programmi devono funzionare correttamente in tutti i casi e *con tutti gli ordinamenti possibili!*
- ❑ I collaudi («test») non sono in grado di garantirlo!
 - “life-testing of ultrareliable software is infeasible (i.e., to quantify 10^{-8} /h failure rate requires more than 10^8 h of testing)”, in: Butler and Finelli, *The Infeasibility of Quantifying the Reliability of Life-Critical Real-Time Software*, IEEE Trans. on Software Eng., 1993
- ❑ \Rightarrow è necessario adottare *pattern* di programmazione concorrente